Homework 1

HW 1-1 Simulate a Function

Describe the models you use, including the number of parameters (at least two models) and the function you use.
GitHub link: https://github.com/sabbellasri/Deep-learning/tree/main/hw1

Models: Three models with added weight decay for regularization have been defined.

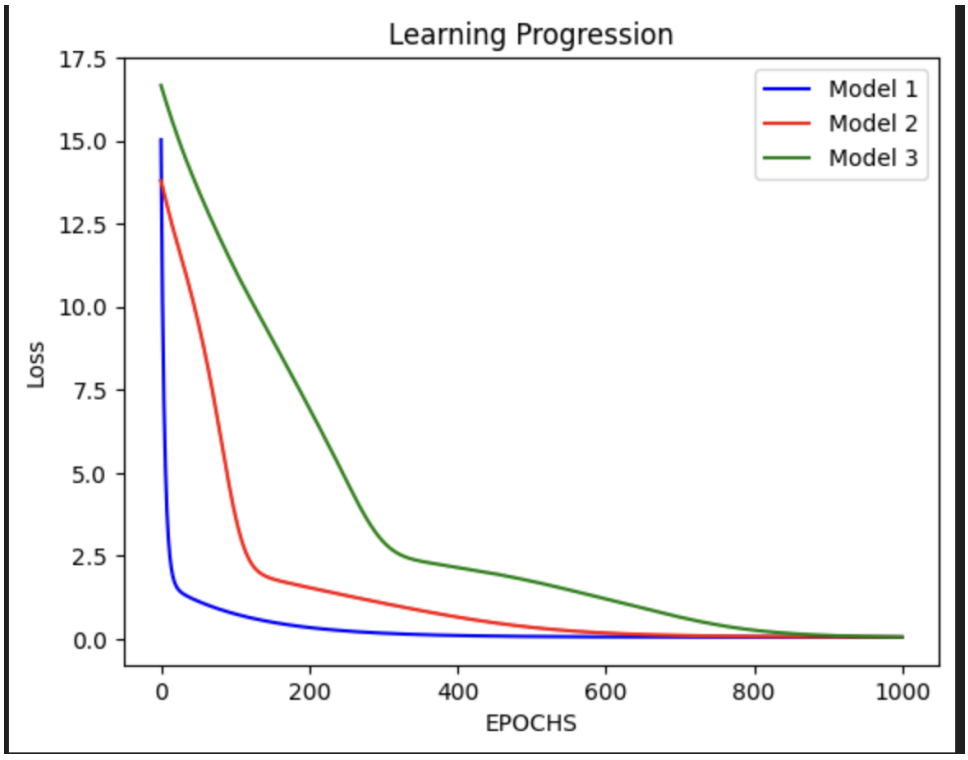
I have implemented three distinct models in accordance with the specified requirements. Each model incorporates a weight decay parameter aimed at enhancing regularization. This parameter introduces a penalty term to the neural network's cost function, effectively leading to weight shrinkage during backpropagation. The first model features 2 dense layers with a total of 571 parameters, utilizing the Mean Squared Error (MSELoss) as the loss function. The RMSProp optimizer is employed with a learning rate of 1e-2, and LeakyRelu serves as the activation function. Similarly, the second model consists of four dense layers and 572 parameters, employing the same MSELoss, RMSProp optimizer with a 1e-2 learning rate, LeakyRelu activation function Lastly, the third model adopts a six dense layer comprising 571 parameters, along with MSELoss, RMSProp optimizer (1e-2 learning rate), LeakyRelu activation function.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 60]             120
            Linear-2                  [-1, 40]           2,440
            Linear-3                  [-1, 20]             820
            Linear-4                   [-1, 1]              21
================================================================
Total params: 3,401
Trainable params: 3,401
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.01
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

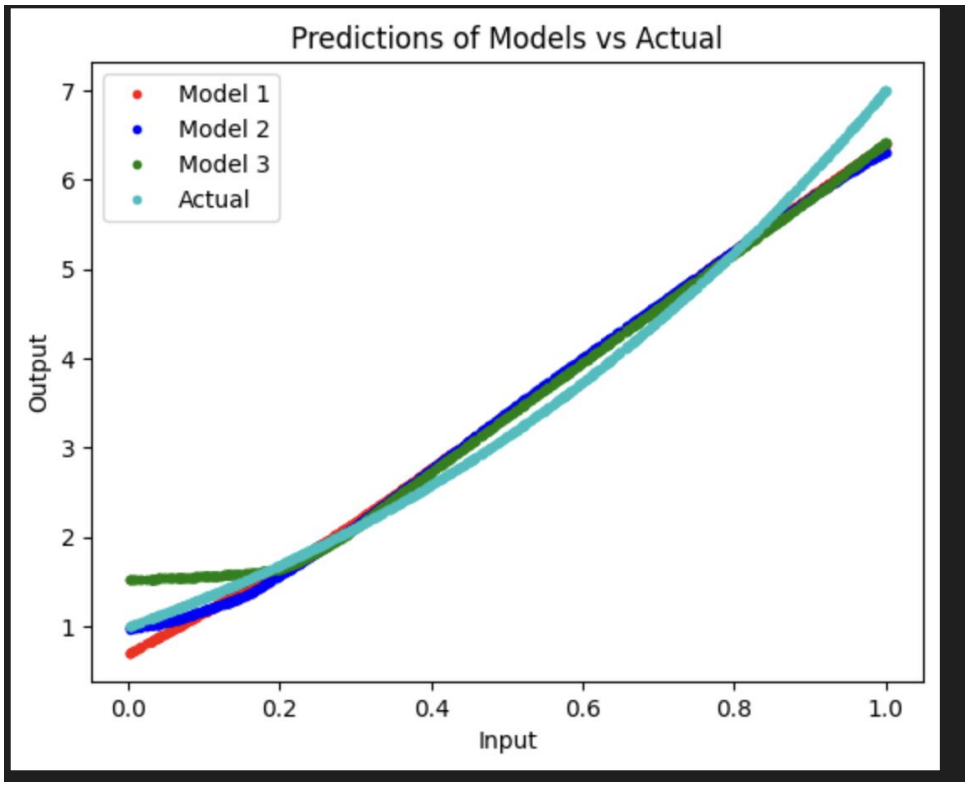
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 250]             500
            Linear-2                   [-1, 1]             251
================================================================
Total params: 751
Trainable params: 751
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.00
Estimated Total Size (MB): 0.00
----------------------------------------------------------------
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                   [-1, 50]             100
            Linear-2                   [-1, 40]           2,040
            Linear-3                   [-1, 30]           1,230
            Linear-4                   [-1, 20]             620
            Linear-5                   [-1, 10]             210
            Linear-6                    [-1, 1]              11
================================================================
Total params: 4,211
Trainable params: 4,211
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.02
Estimated Total Size (MB): 0.02
----------------------------------------------------------------
```

Loss VS epocs graph for the 3 models



Predicted function curve of all the models



Result

Models 1 and 2 demonstrate faster convergence compared to Model 3, reaching lower loss values and effectively learning the underlying function. The graphical representation highlights that Models 1 and 2 outperform Model 3, emphasizing the impact of layer count on the models' learning capabilities. The greater number of layers in Models 1 and 2 contributes to their faster and more effective learning process.

HW 1-1 Train on Actual Tasks

The below models are trained on the MNIST dataset. Training Set: 60,000 Testing set: 10,000

Model 1: CNN (LeNet)

- 2D convolution layer: apply a 2D max pooling -> ReLu
- 2D convolution layer: apply a 2D max pooling -> ReLu
- 2D Dense Layer: ReLu
- 2D Dense Layer: ReLu
- 2D Dense Layer: ReLu

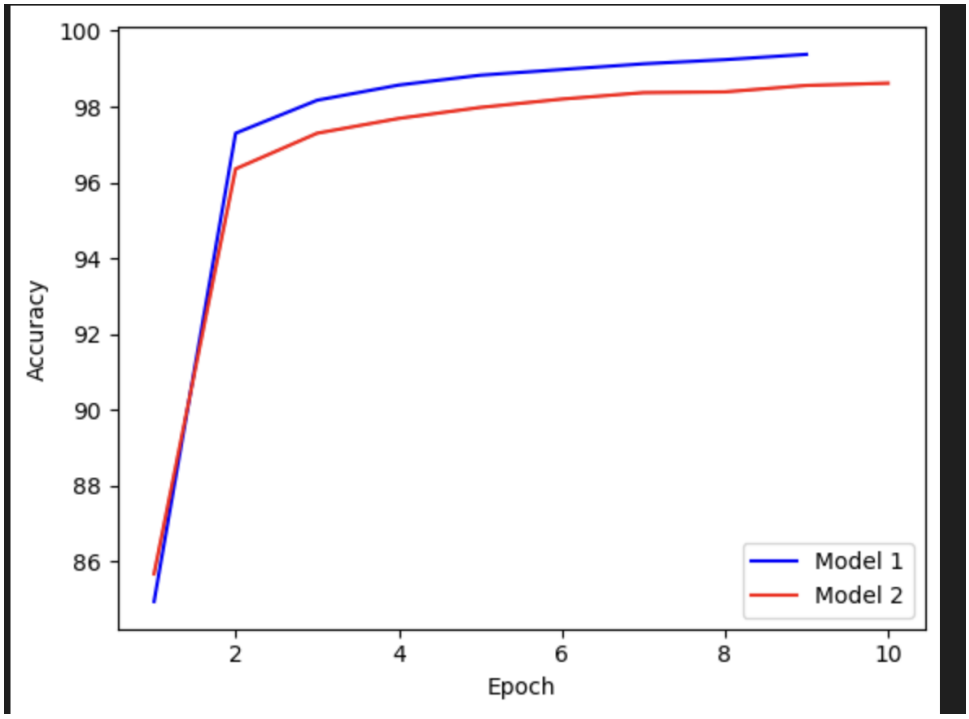
Model 2: CNN (custom)

- 2D convolution layer: ReLu
- 2D convolution layer: apply a 2D max pooling -> ReLu -> Dropout
- 2D convolution layer: apply a 2D max pooling -> ReLu -> Dropout
- 2D Dense Layer: ReLu -> Dropout
- 2D Dense Layer: Log_softmax (Output)

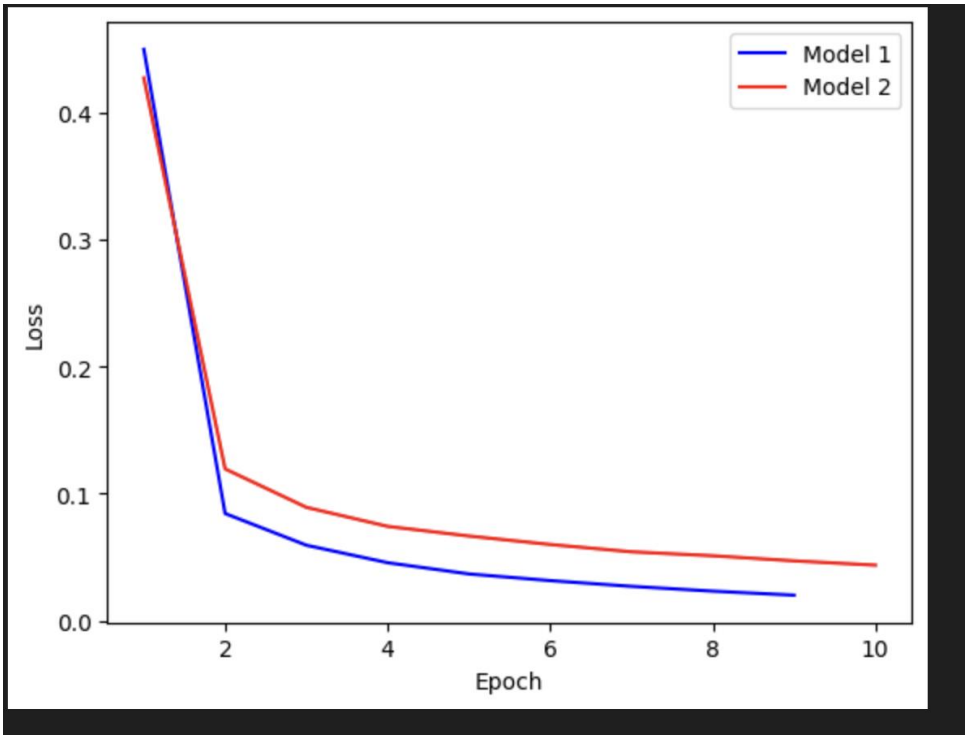
Hyperparameters

- Learning Rate: 0.01
- Momentum: 0.9
- Optimizer: Stochastic Gradient Descent
- batch_size = 64
- epochs = 10
- Loss: Cross Entropy

Below is the training loss for Model 1 and Model 2



Below is the training accuracy for Model 1 and Model 2



Result

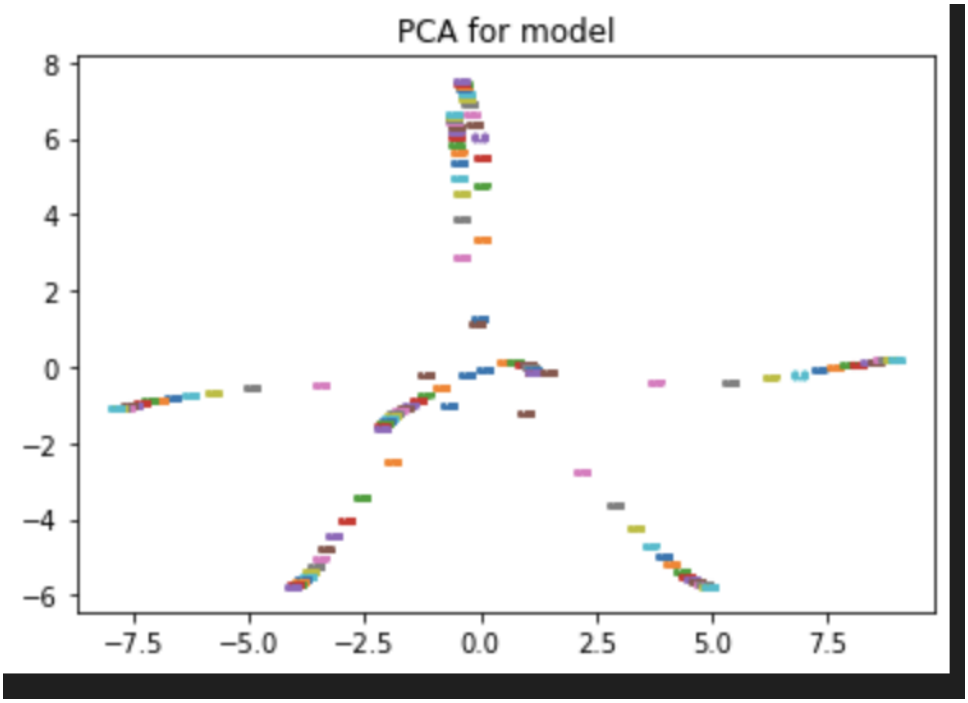
Model 1 outperforms Model 2, exhibiting lower loss and superior performance. Its architecture is an optimized convolutional neural network (CNN) model, drawing inspiration from the well-established LeNet structure. Across the range of epochs, Model 1 consistently surpasses a custom-built CNN model. This trend extends to accuracy, with Model 1 achieving higher training accuracy compared to Model 2, emphasizing its superior predictive capabilities.

HW 1-2 Visualize the optimization process

Collect the weights every 3 epochs, and train 8 times. Reduce the dimension of weights to 2 by PCA.

The below model is trained on the MNIST dataset.
Training Set: 60,000 Testing set: 10,000

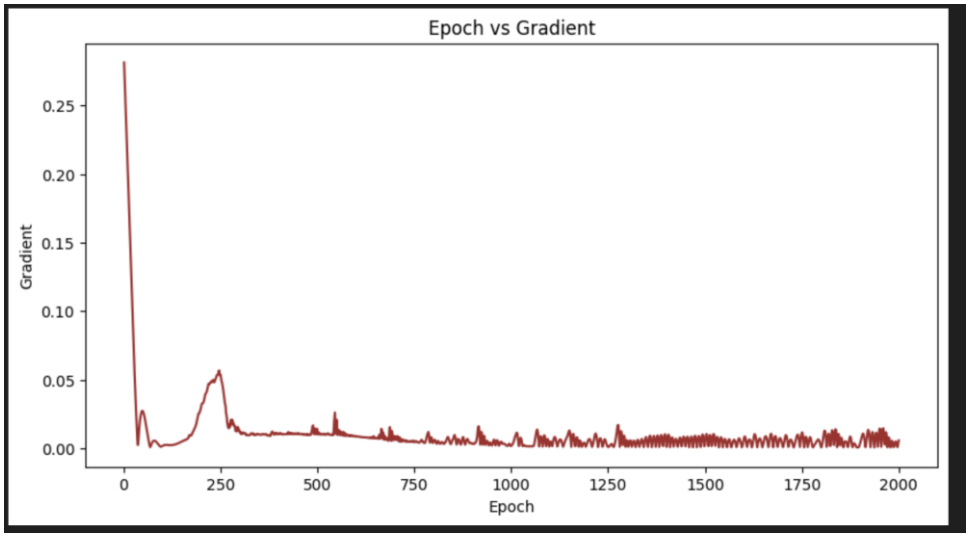
- 3 Dense Layers
- Loss Function: Cross Entropy Loss
- Optimizer: Adam
- Learning Rate: 0.0004
- Batch size: 1000
- Activation Function: ReLu



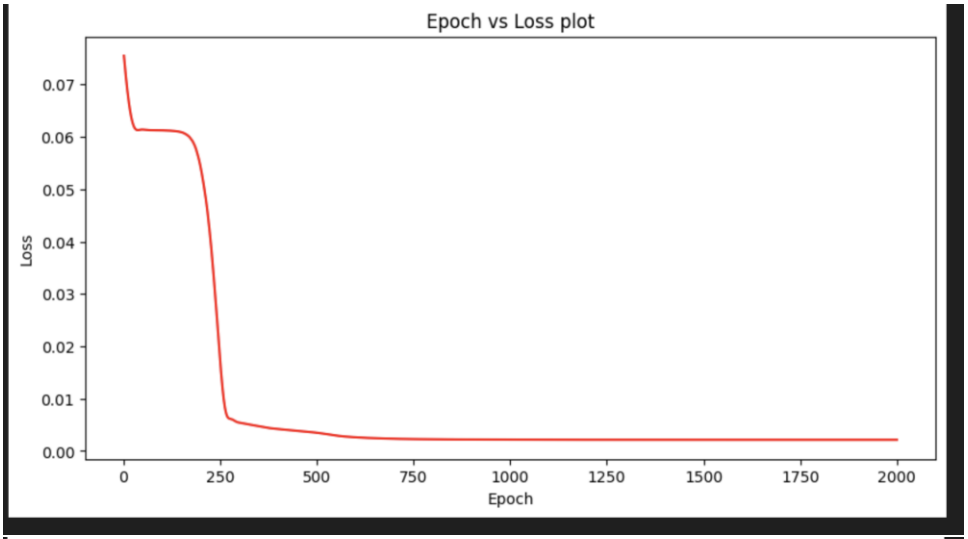
The depicted graph showcases the weights obtained through the PCA algorithm applied to the collected data. The reduction in dimensionality is evident, illustrating the transformation from the original set of 8240 parameters or weights for each model. The models underwent training eight times for 45 epochs, and the resulting graph reflects the impact of PCA on the weight representations. HW 1-2 Observe gradient norm during training

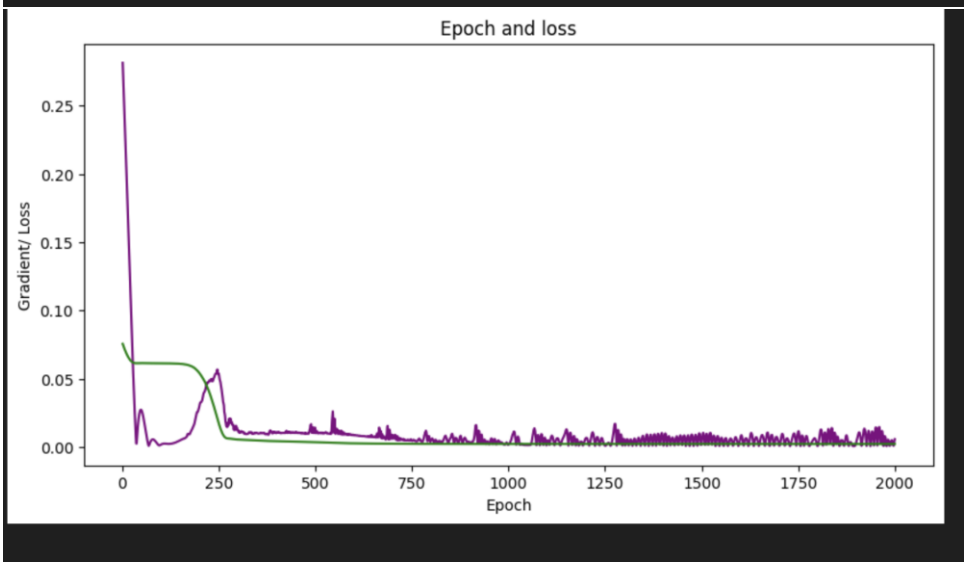
The function sin(5*pi*x) / 5*pi*x has been reused to calculate the gradient norm and the loss. I have trained the model on epochs rather than iterations as the input to the model is already a small size.

Below is a graph for gradient norm across the epochs.



Below is a graph for loss across the epochs





Result

The model has successfully undergone training and reached convergence. After the initial 100 epochs, a subtle rise in the gradient is noticeable, mirrored by a plateau in the loss graph. This plateauing effect in the loss graph suggests a slower rate of decrease, eventually stabilizing around 250 epochs.

HW 1-3 Can network fit random labels?

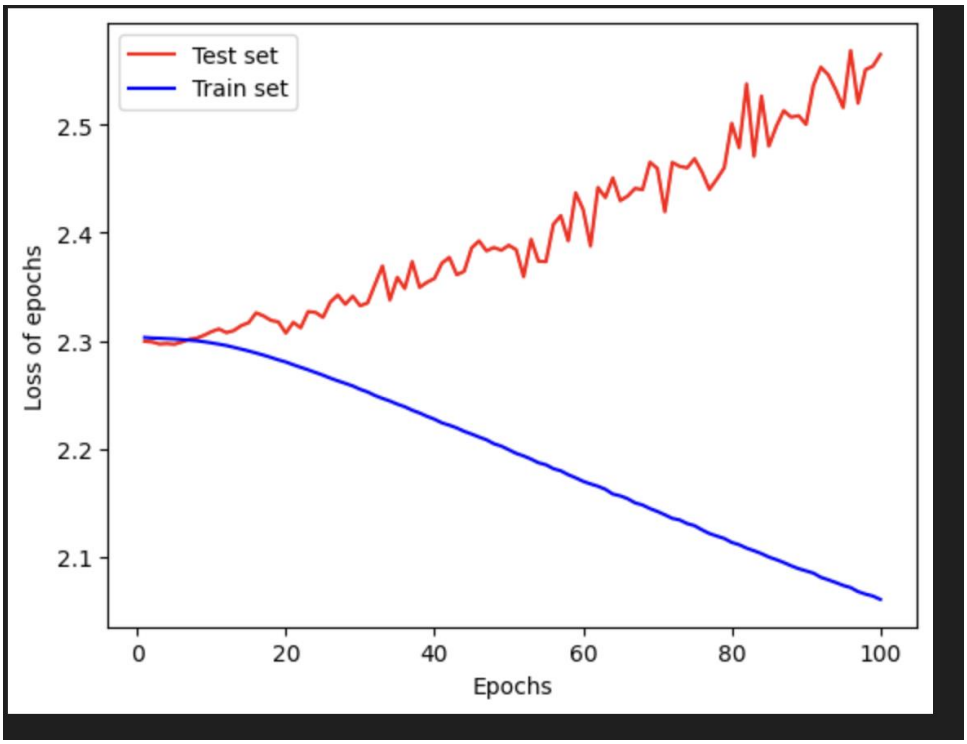
The below model is trained on the MNIST dataset. Training Set: 60,000 Testing set: 10,000

Model 1: CNN (LeNet)

- 2D convolution layer: apply a 2D max pooling -> ReLu
- 2D convolution layer: apply a 2D max pooling -> ReLu
- 2D Dense Layer: ReLu
- 2D Dense Layer: ReLu

Hyperparameters

- Learning Rate: 0.0001
- Optimizer: Adam
- train_batch_size = 100
- test_batch_size = 100
- epochs = 100
- loss function: Cross Entropy Loss



The model has been trained on randomly assigned labels, resulting in a slow training process. As the epochs progress, the model attempts to memorize these random labels, leading to a reduction in the loss. Notably, the test loss shows a continuous increase with the growth of epochs, accompanied by a

gradual decrease in the training loss. The depicted graph confirms this phenomenon, illustrating a widening gap between the training and test losses as the number of epochs increases.

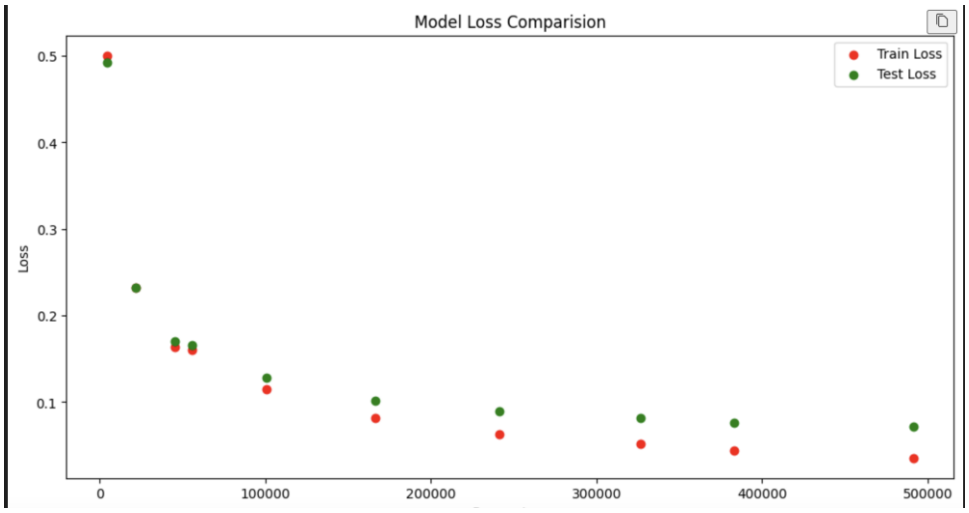
HW 1-3 Number of parameters vs Generalization

The below model is trained on the MNIST dataset.
Training Set: 60,000 Testing set: 10,000

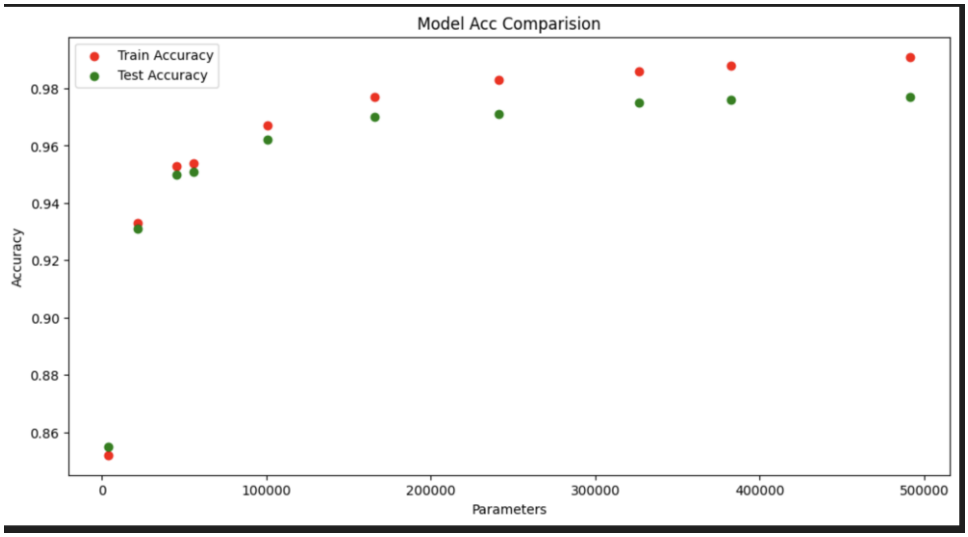
- 3 Dense Layers
- Loss Function: Cross Entropy Loss
- Optimizer: Adam
- Learning Rate: 1e-3
- Batch size: 50
- Activation Function: ReLu

We vary the size of the models by approximately doubling the inputs and outputs at every dense layer, this increases the number of parameters for training.

Graph for Loss comparison for the various models



Graph for Accuracy comparison for the various models



The presented graphs highlight that as the number of parameters increases, the disparity between the train and test loss/accuracy becomes more pronounced. The test loss demonstrates an earlier plateau compared to the training loss or accuracy.

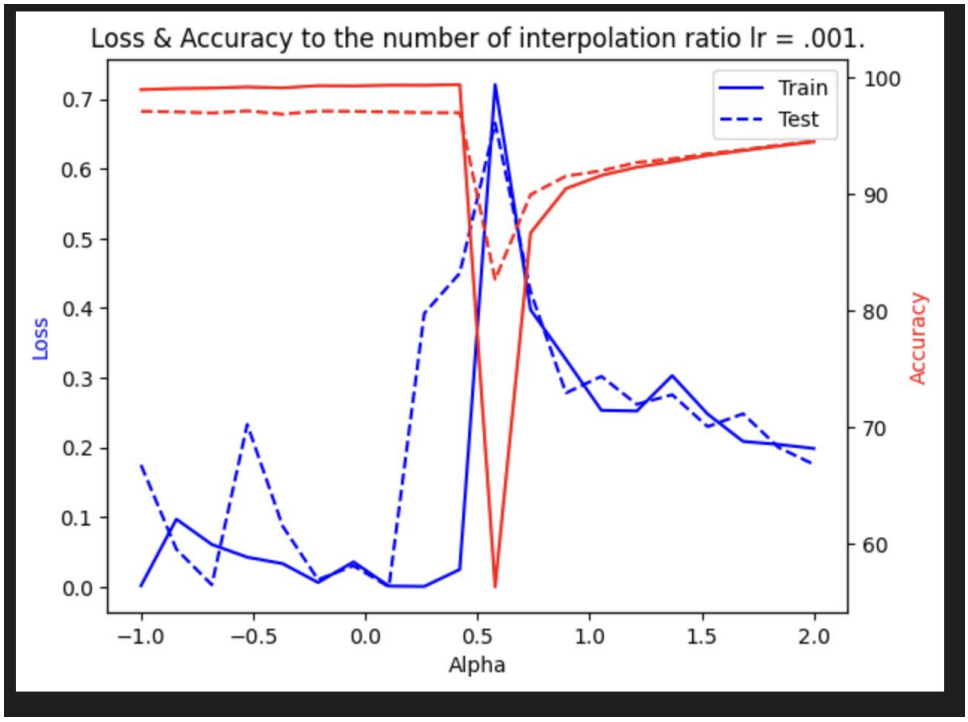
This phenomenon is attributed to overfitting, wherein the model, equipped with a greater number of parameters, tends to memorize the training data excessively. While this may enhance accuracy and reduce loss during training, it becomes crucial to minimize the gap between train and test loss/accuracy to mitigate overfitting. Unfortunately, due to computational constraints, further augmentation of parameters was not feasible; nonetheless, the observed trend is indicative in the provided graphs.

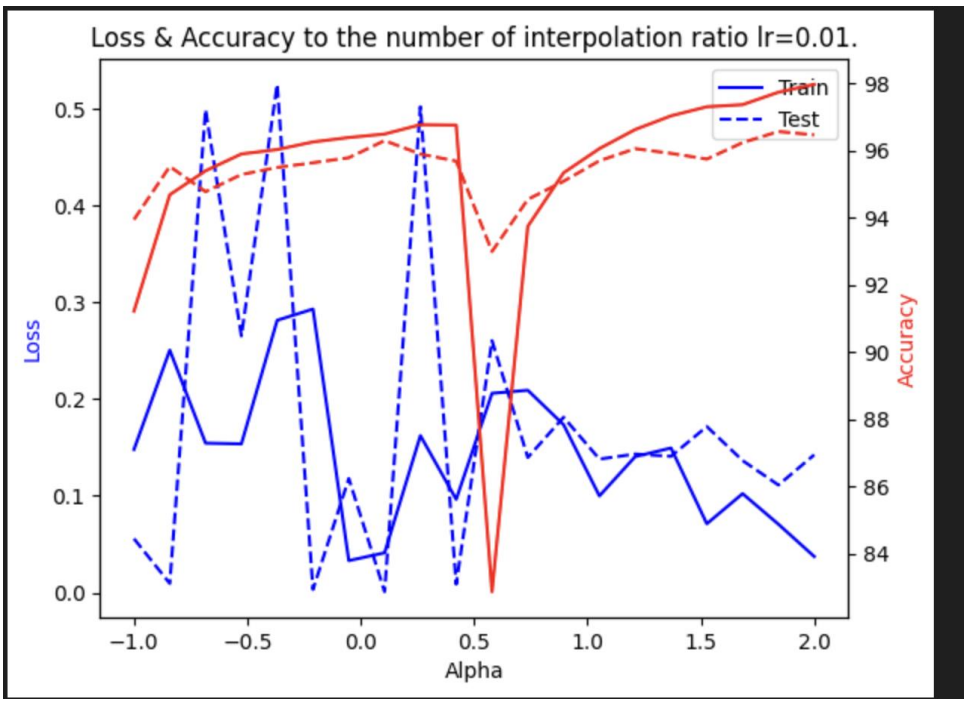
HW 1-3 Flatness vs Generalization Part 1

The below model is trained on the MNIST dataset. Training Set: 60,000 Testing set: 10,000

- 3 Linear layers
- Loss Function: Cross Entropy Loss
- Optimizer: SGD
- Learning Rate: 1e-3, 1e -2
- Batch size: 100, 500
- Activation Function: ReLu

- The training process involves models trained with diverse batch sizes, and their weights are collected. Utilizing the interpolation formula "(1-alpha) * batch1_param + alpha * batch2_param," we calculate the interpolation ratio for these weights. Subsequently, 50 models are generated using the updated weight values determined by the interpolation ratio.
    - The loss and accuracy of these new models are then recorded and visualized in a comprehensive graph, providing insights into the performance of models trained with varying batch sizes. The learning rate employed for this process is set at 1e-2





Learning Rate 1e-3

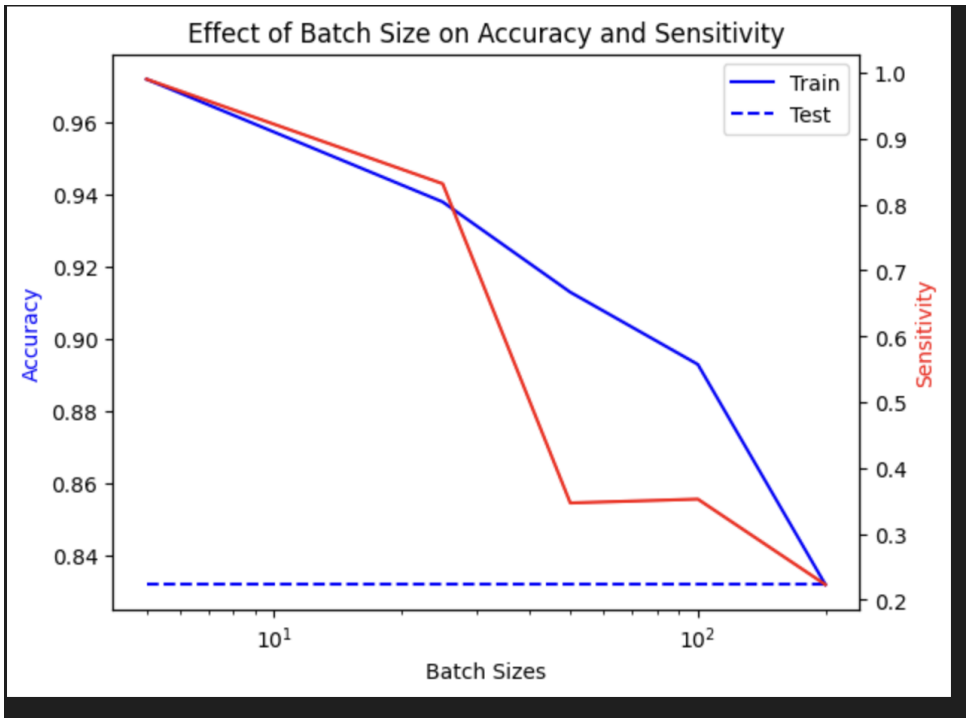
The accuracy of both the test and train sets exhibits a noticeable decline around an alpha value of 1.5 for models with diverse learning rates. The upward trend in accuracy becomes more pronounced as the alpha value reaches 1.5. Here, alpha represents the interpolation ratio determined by the formula "(1-alpha) * batch1_param + alpha * batch2_param."

In summary, machine learning algorithms demonstrate a form of interpolation between data points. However, when the number of parameters exceeds the available data, the models tend to memorize the data and interpolate between these memorized points.

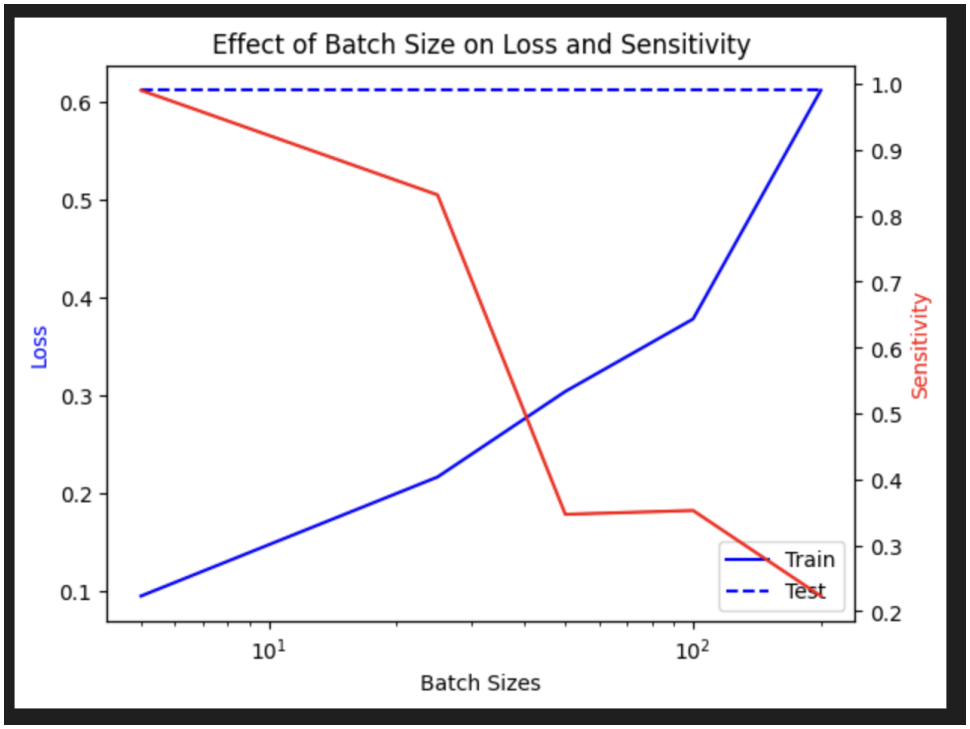
Flatness vs Generalization – Part 2

The below model is trained on the MNIST dataset. Training Set: 60,000 Testing set: 10,000

- 3 Linear layer
- Loss Function: Cross Entropy Loss
- Optimizer: SGD
- Learning Rate: 1e-3
- batchSizes = [5, 25, 50, 100, 200]
- Activation Function: ReLu



- Note: In the graphs shown the legend is displayed incorrectly, the blue line refers to the sensitivity and the red/blue lines refer to the model Accuracy and losses respectively.
- X -axis refers to batch size and the Y-axis for the graph on top refers to loss.
- Y-axis for the graph below refers to accuracy.



From the above graphs we can see that the sensitivity decreases with increase in batch size. The sensitivity peaks with an optimum batch size of around 4000 and then continues to reduce with drop in accuracy and increase in loss value. We can conclude that the network becomes less sensitive as the batch size increases beyond 5000 epochs.